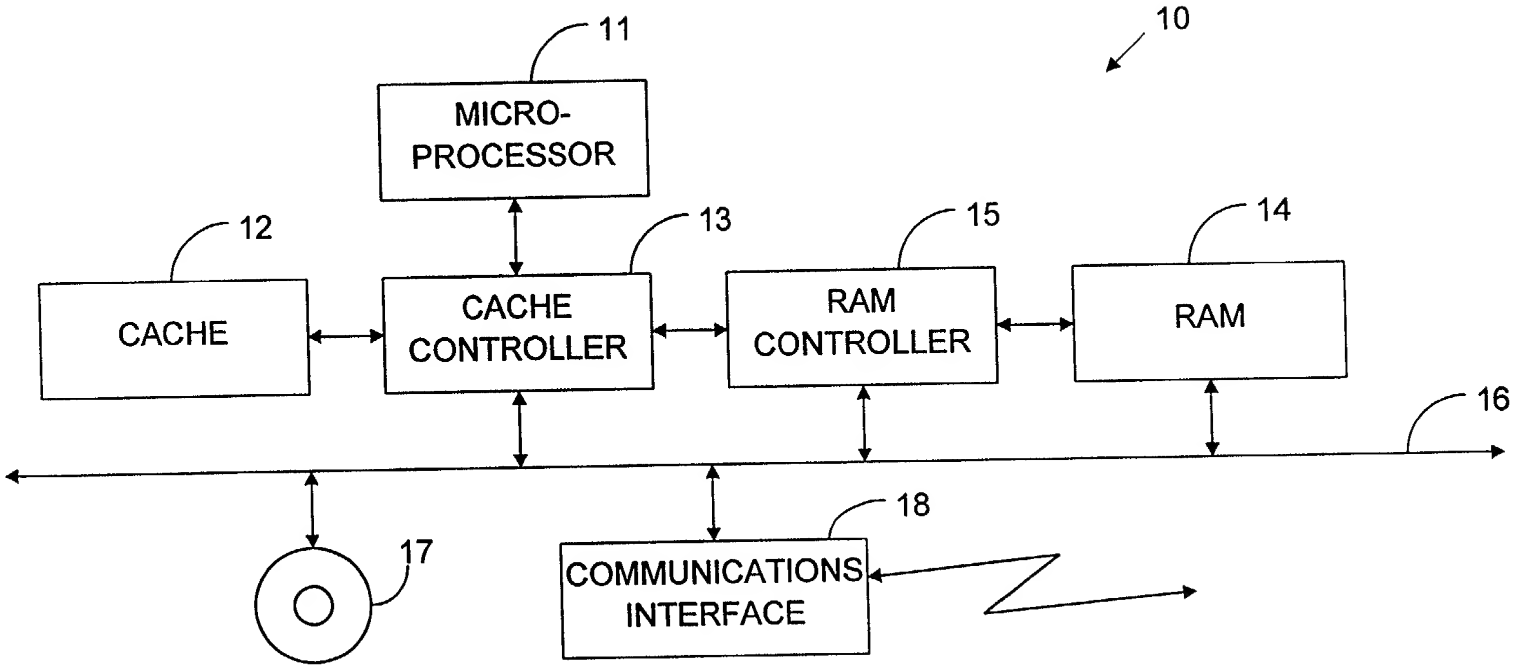
10
11
MICRO-
PROCESSOR
12
CACHE
13
CACHE
CONTROLLER
15
RAM
CONTROLLER
14
RAM
16
17
18
COMMUNICATIONS
INTERFACE

FIG. 1

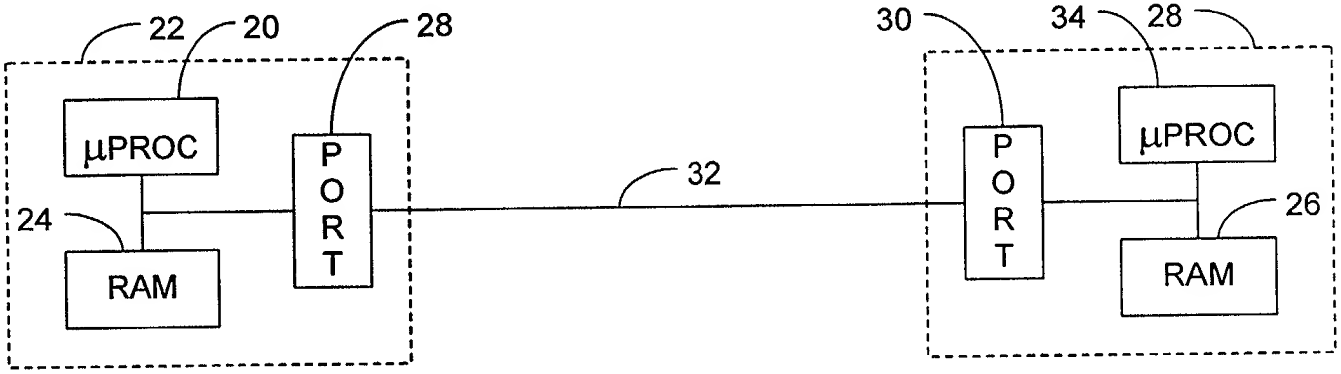
22
20
28
30
34
28
µPROC
PORT
32
PORT
µPROC
24
RAM
26
RAM

FIG. 2

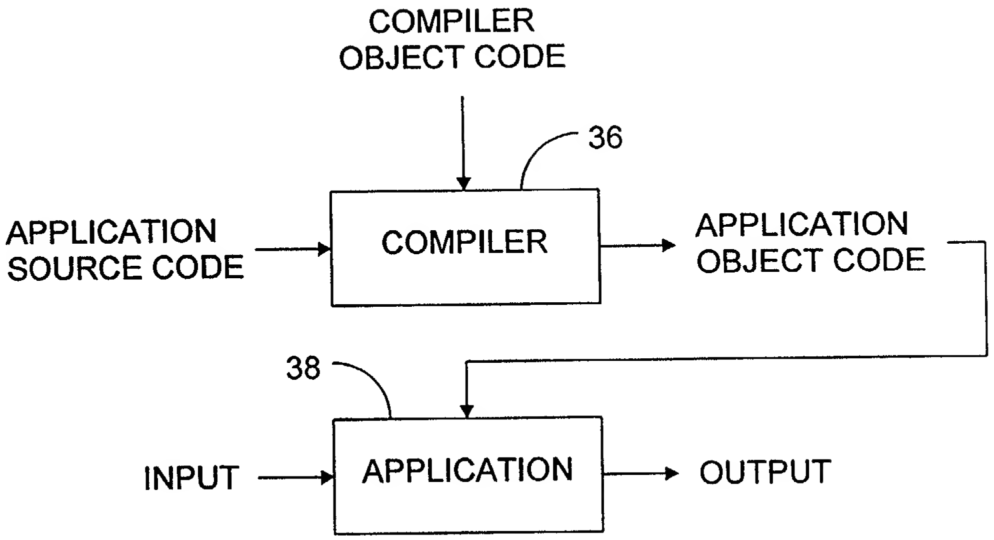
COMPILER
OBJECT CODE
36
APPLICATION
SOURCE CODE
COMPILER
APPLICATION
OBJECT CODE
38
INPUT
APPLICATION
OUTPUT

FIG. 3

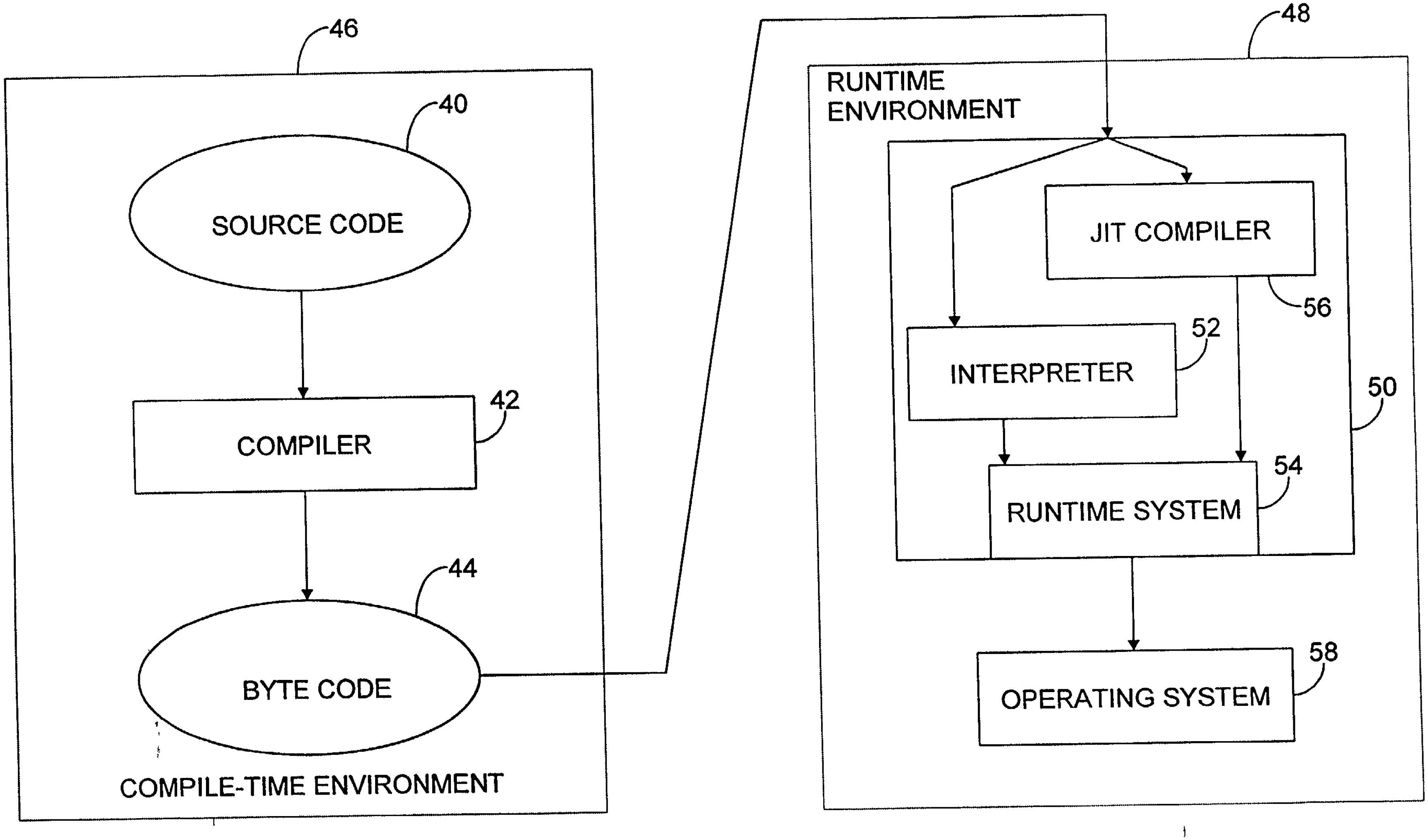
46
40
SOURCE CODE
COMPILER
42
44
BYTE CODE
COMPILE-TIME ENVIRONMENT
48
RUNTIME
ENVIRONMENT
JIT COMPILER
56
52
INTERPRETER
50
54
RUNTIME SYSTEM
58
OPERATING SYSTEM

FIG. 4

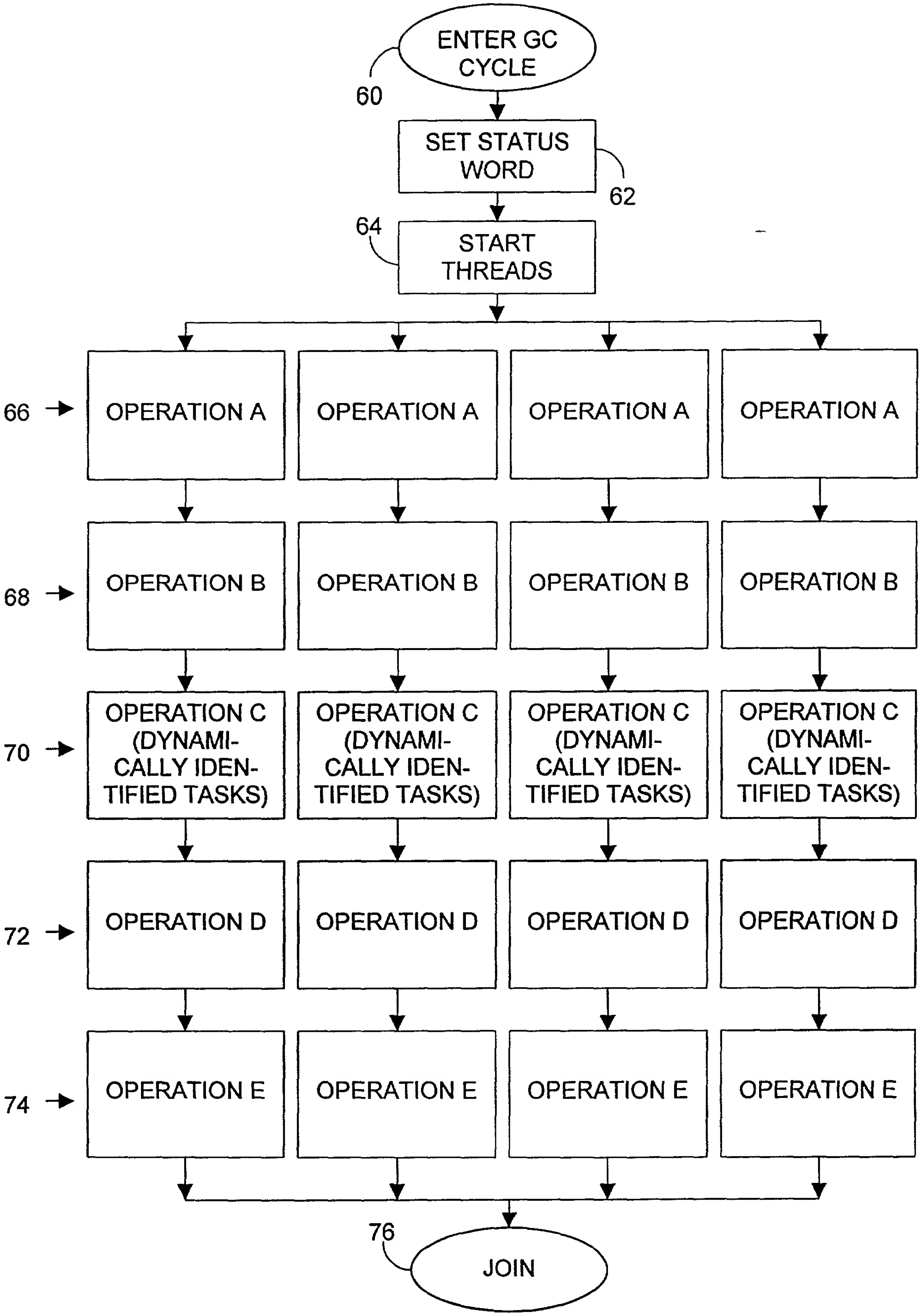
ENTER GC CYCLE
60
SET STATUS WORD
62
64
START THREADS
66
OPERATION A
OPERATION A
OPERATION A
OPERATION A
68
OPERATION B
OPERATION B
OPERATION B
OPERATION B
70
OPERATION C (DYNAMICALLY IDENTIFIED TASKS)
OPERATION C (DYNAMICALLY IDENTIFIED TASKS)
OPERATION C (DYNAMICALLY IDENTIFIED TASKS)
OPERATION C (DYNAMICALLY IDENTIFIED TASKS)
72
OPERATION D
OPERATION D
OPERATION D
OPERATION D
74
OPERATION E
OPERATION E
OPERATION E
OPERATION E
76
JOIN

FIG. 5

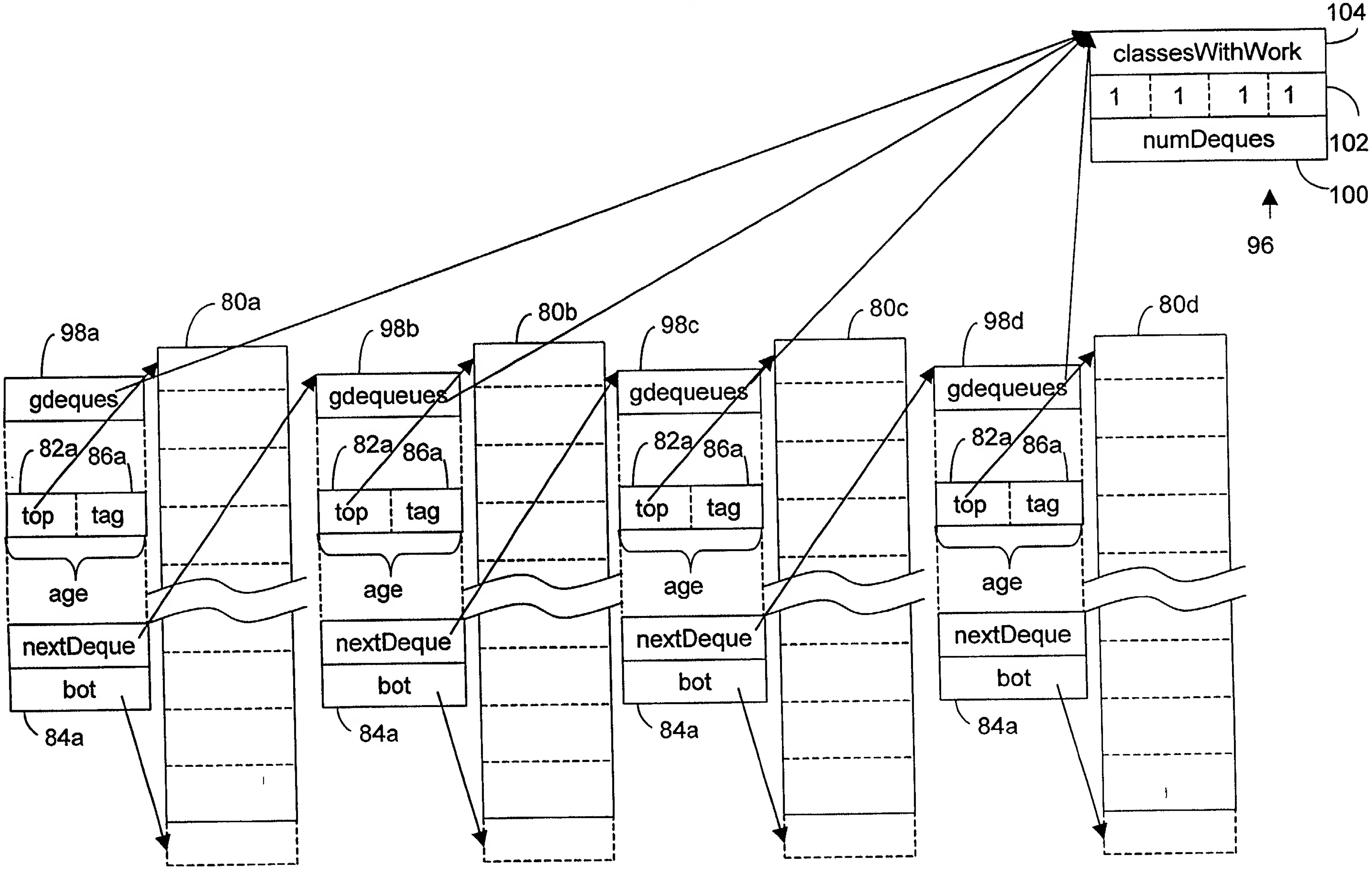
104
classesWithWork
1
1
1
1
numDeques
102
100
96
98a
gdeques
82a
86a
top
tag
age
nextDeque
bot
84a
80a
98b
gdequeues
82a
86a
top
tag
age
nextDeque
bot
84a
80b
98c
gdequeues
82a
86a
top
tag
age
nextDeque
bot
84a
80c
98d
gdequeues
82a
86a
top
tag
age
nextDeque
bot
84a
80d

FIG. 6

```
java_lang_Object* popTop ()
{
        oldAge = age;
        localBot = bot;
        if (localBot <= oldAge.top)
            return NULL;
        task = deq[oldAge.top];
        newAge = oldAge;
        newAge.top++;
        cas(age, oldAge, newAge); /*atomic compare-and-swap*/
        if(oldAge == newAge)
             return task;
        return NULL;
}
```

FIG. 7

```
void pushBottom(java_lang_Object* task)
{
        localBot = bot;
        deq[localBot] = task;
        localBot++;
        bot = localBot;
}
```

FIG. 8

```
java_lang_Object* popBottom()
{
        localBot = bot;
        if (localBot == 0)
          return NULL;
        localBot--;
        bot = localBot;
        task = deq[localBot];
        oldAge = age;
        if (localBot > oldAge.top)
          return task;
        bot = 0;
        newAge.top = 0;
        newAge.tag = oldAge.tag + 1;
        if (localBot == oldAge.top) {
          cas(age, oldAge, newAge)
          if (oldAge == newAge)
             return task;
        }
        age = newAge;
        return NULL;
}
```

FIG. 9

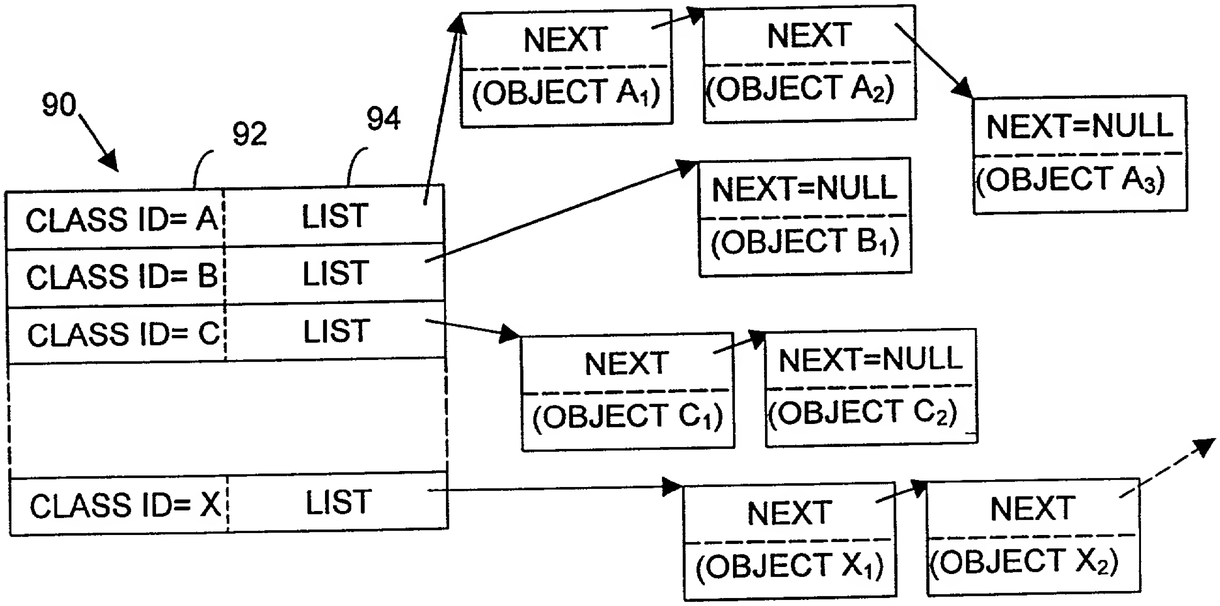

FIG. 10

```
java_lang_Object *dequeFindWork(localDeque *dq) {
    java_lang_Object *result = findWorkHelper(dq);
    globalDeques *gdqs = dq->gdeques;
    if (result == NULL) {
        mark_self_inactive(dq->index, &gdqs->statusBitmap); /* You have no work */
    }
    while (result == NULL) {
        if (!gdqs->statusBitmap) return NULL;  /* No one has any work. Terminate. */
        poll(NULL, NULL, 0);
        if (checkForWork(dq)) { /* You don't have any work, but there is some either
                    on the overflow queue, or in another thread's work
                    queue */
            mark_self_active(dq->index, &gdqs->statusBitmap); /* Looking for work */
            result = findWorkHelper(dq);
            if (result == NULL) {
                  mark_self_inactive(dq->index, &gdqs->statusBitmap);
            }
        }
    }
    return result;
}
```

```
Java_lang_Object *findWorkHelper(localDeque *dq) {
    java_lang_Object *task = findWorkInOverflowList(dq);
    if (task == NULL) {
        task = stealWork(dq);
    }
    return task;
}
```

FIG. 11

```
static java_lang_Object *stealWork(localDeque *dq) {
    globalDeques *gdqs = dq->gdeques;
    int degree = gdqs->numDeques;
    int iterations = 2 * degree;
    int i = 0;
    while (i++ < iterations) {
        localDeque *dqToSteal = pickQueueToStealFrom(gdqs, dq);
        if (dqToSteal->bot > dqToSteal->age.top) {
            java_lang_Object *task = (java_lang_Object *)popTop(dqToSteal);
            if(!task) poll(NULL, NULL, 0);
            else return task;
        }
    }
    return NULL;
}
```

FIG. 12

```
static bool_t checkForWork(localDeque *dq) {
    globalDeques *gdqs = dq->gdeques;
    return gdqs->classesWithWork || peekDeque(dq);
}
```

```
static bool_t peekDeque(localDeque *dq) {
   globalDeques *gdqs = dq->gdeques;
   int degree = gdqs->numDeques;
   int i;
   for (i = 0; i < 2 * degree; i++) {
      localDeque *dqToPeek = pickQueueToStealFrom(gdqs, dq);
      if (dqToPeek->bot > dqToPeek->age.top) {
        return TRUE;
      }
   }
   return FALSE;
}
```

FIG. 13